import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import LabelEncoder, StandardScaler

from sklearn.feature\_selection import SelectKBest, f\_classif

from imblearn.over\_sampling import SMOTE

import matplotlib.pyplot as plt

*# Load the dataset*

classification\_data = pd.read\_csv('/content/data set.csv')

*# Preprocessing*

classification\_data = classification\_data.dropna(axis=1, how='all') *# Drop empty columns*

X = classification\_data.iloc[:, :-1] *# All columns except the last one as features*

y = classification\_data.iloc[:, -1] *# Last column as target*

*# Encode non-numeric target and features*

le = LabelEncoder()

if y.dtype == 'object':

y = le.fit\_transform(y)

for col in X.select\_dtypes(include=['object']).columns:

X[col] = le.fit\_transform(X[col].astype(str))

*# Normalize features*

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

*# Train-test split*

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.2, random\_state=42)

*# Handle class imbalance using SMOTE*

smote = SMOTE(random\_state=42)

X\_resampled, y\_resampled = smote.fit\_resample(X\_train, y\_train)

*# Feature Selection: Select top 10 features based on ANOVA F-statistic*

k\_best = SelectKBest(score\_func=f\_classif, k=10)

X\_train\_kbest = k\_best.fit\_transform(X\_resampled, y\_resampled)

X\_test\_kbest = k\_best.transform(X\_test)

*# Models*

underfit\_model = LogisticRegression(max\_iter=500, penalty='l2', C=0.01, random\_state=42) *# Underfitting model*

balanced\_model = RandomForestClassifier(random\_state=42) *# To be tuned*

overfit\_model = DecisionTreeClassifier(max\_depth=None, min\_samples\_split=2, random\_state=42) *# Overfitting model*

*# Hyperparameter tuning for balanced model*

param\_grid = {

'n\_estimators': [50, 100, 200, 300],

'max\_depth': [5, 10, 15, 20, None],

'min\_samples\_split': [2, 5, 10, 15],

'min\_samples\_leaf': [1, 2, 4],

'max\_features': ['sqrt', 'log2', None]

}

random\_search = RandomizedSearchCV(

estimator=balanced\_model,

param\_distributions=param\_grid,

n\_iter=50, *# Number of parameter combinations to try*

cv=5, *# 5-fold cross-validation*

scoring='accuracy',

n\_jobs=-1,

verbose=2,

random\_state=42

)

*# Perform RandomizedSearchCV*

random\_search.fit(X\_train\_kbest, y\_resampled)

balanced\_model = random\_search.best\_estimator\_ *# Best model after tuning*

print("Best Parameters for Balanced Model:", random\_search.best\_params\_)

*# Train and evaluate all models*

models = {

"Underfitting (Logistic Regression)": underfit\_model,

"Balanced (Random Forest)": balanced\_model,

"Overfitting (Unconstrained Decision Tree)": overfit\_model

}

train\_accuracies, test\_accuracies = [], []

for name, model in models.items():

model.fit(X\_train\_kbest, y\_resampled)

train\_accuracies.append(accuracy\_score(y\_resampled, model.predict(X\_train\_kbest)))

test\_accuracies.append(accuracy\_score(y\_test, model.predict(X\_test\_kbest)))

*# Visualize train vs test accuracy*

plt.figure(figsize=(10, 6))

bars\_train = plt.bar([x + 0.2 for x in range(len(models))], train\_accuracies, width=0.4, label="Train Accuracy", color='skyblue')

bars\_test = plt.bar([x - 0.2 for x in range(len(models))], test\_accuracies, width=0.4, label="Test Accuracy", color='orange')

*# Annotate bars*

for i, acc in enumerate(train\_accuracies):

plt.text(i + 0.2, acc + 0.02, f'{acc:.2f}', ha='center')

for i, acc in enumerate(test\_accuracies):

plt.text(i - 0.2, acc + 0.02, f'{acc:.2f}', ha='center')

plt.xticks(range(len(models)), models.keys(), rotation=15)

plt.ylabel("Accuracy")

plt.title("Underfitting vs Overfitting vs Balanced Model (Improved)")

plt.legend()

plt.ylim(0, 1.1)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.show()

*# Bias-Variance Trade-Off (Decision Tree Depth vs Error)*

depths = range(1, 21)

train\_errors, test\_errors = [], []

for depth in depths:

model = DecisionTreeClassifier(max\_depth=depth, min\_samples\_split=10, random\_state=42)

model.fit(X\_train\_kbest, y\_resampled)

train\_errors.append(1 - accuracy\_score(y\_resampled, model.predict(X\_train\_kbest)))

test\_errors.append(1 - accuracy\_score(y\_test, model.predict(X\_test\_kbest)))

*# Smooth the validation curve using a rolling average*

test\_errors\_smooth = pd.Series(test\_errors).rolling(window=3, center=True).mean()

plt.figure(figsize=(10, 6))

plt.plot(depths, train\_errors, label="Training Error", marker="o", color='blue')

plt.plot(depths, test\_errors\_smooth, label="Validation Error (Smoothed)", marker="o", color='orange')

plt.title("Bias-Variance Trade-Off")

plt.xlabel("Model Complexity (Tree Depth)")

plt.ylabel("Error")

plt.legend()

plt.grid(True, linestyle='--', alpha=0.7)

plt.show()

*# Confusion Matrix*

from sklearn.metrics import ConfusionMatrixDisplay

for name, model in models.items():

ConfusionMatrixDisplay.from\_estimator(model, X\_test\_kbest, y\_test, cmap='Blues')

plt.title(f"Confusion Matrix for {name}")

plt.show()

*# ROC Curve and AUC*

from sklearn.metrics import roc\_curve, auc

for name, model in models.items():

if hasattr(model, "predict\_proba"): *# Ensure the model can produce probabilities*

y\_prob = model.predict\_proba(X\_test\_kbest)[:, 1]

fpr, tpr, \_ = roc\_curve(y\_test, y\_prob)

roc\_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label=f'{name} (AUC = {roc\_auc:.2f})')

plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")

plt.title("ROC Curve")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.legend(loc="lower right")

plt.grid(alpha=0.7)

plt.show()

*# Learning Curve*

from sklearn.model\_selection import learning\_curve

train\_sizes, train\_scores, test\_scores = learning\_curve(

balanced\_model, X\_train\_kbest, y\_resampled, cv=5, scoring='accuracy', n\_jobs=-1, train\_sizes=np.linspace(0.1, 1.0, 10)

)

train\_mean = np.mean(train\_scores, axis=1)

test\_mean = np.mean(test\_scores, axis=1)

plt.plot(train\_sizes, train\_mean, label="Training Accuracy", marker='o', color='blue')

plt.plot(train\_sizes, test\_mean, label="Validation Accuracy", marker='o', color='orange')

plt.title("Learning Curve (Balanced Random Forest)")

plt.xlabel("Training Set Size")

plt.ylabel("Accuracy")

plt.legend()

plt.grid(alpha=0.7)

plt.show()

Fitting 5 folds for each of 50 candidates, totalling 250 fits

Best Parameters for Balanced Model: {'n\_estimators': 100, 'min\_samples\_split': 2, 'min\_samples\_leaf': 4, 'max\_features': 'sqrt', 'max\_depth': 5}













